# ÔN LẠI KIẾN THỨC VỀ JAVASCRIPT

## Arrow function trong ES6

Là một phương thức rút gọn dành cho hàm, sử dụng mũi tên “=>”

Ví dụ:

* Đây là một hàm bình thường để in ra một đoạn console

*function logger (log) {*

*console.log(log);*

*}*

*logger("Hello");*

* Đây là sử dụng arrow function để rút gọn

*const logger = (log) => {*

*console.log(log);*

*}*

*logger("Hello");*

* Hoặc có thể rút gọn dấu ngoặc đơn nếu chỉ có 1 tham số

*const logger = log => {*

*console.log(log);*

*}*

*logger("Hello");*

* Đây là dạng tối giản hơn: Rút gọn dấu ngoặc nhọn và return, mặc định sau dấu “=>” là return, và thường sử dụng cho 1 lệnh đơn giản

*const logger = (log) => console.log(log);*

*logger("Hello");*

* Cũng như cách trên nhưng là một khối lệnh, khối lệnh này in ra object a và b, sẽ có 2 phép gán nên phải sử dụng ngoặc đơn

*const logger = (a, b) => ({a: a, b: b});*

*console.log(logger(2, 3));*

Tuy nhiên thì arrow function không có context riêng. Context được hiểu như là ngữ cảnh hay phạm vi của đối tượng. Xem ví dụ sau để hiểu

Ví dụ:

* Tạo một object, có thể thấy được khi gọi *this.name* thì this ở đây được hiểu như là đối tượng gọi đến getName (tức là course), nếu sử dụng function ở đây thì sẽ in ra được từ “Javacript”

*const course = {*

*name: "Javascript",*

*getName: function () {*

*return this.name;*

*}*

*}*

*console.log(course.getName());*

* Nhưng nếu sử dụng arrow function thì sẽ in ra undefined vì arrow function không có context riêng nên không thể gọi được name, vậy nên trong trường hợp này không sử dụng arrow function

*const course = {*

*name: "Javascript",*

*getName: () => {*

*return this.name;*

*}*

*}*

*console.log(course.getName());*

## Module trong ES6

Là việc tác các thành phần quan trọng ra các nơi riêng, tập hợp các file nhỏ lẻ

Import là nạp vào, Export là xuất ra. Để sử dụng import và export buộc file js phải có kiểu dữ liệu là module. VD: <script type="module" src="./script.js"></script>

Mỗi file js chỉ được phép export default 1 lần duy nhất, có thể có nhiều export thông thường

## Enhanced object literals trong ES6

* Ví dụ: Tạo một đối tượng như sau

*var name = 'JavaScript'*

*var price = 1000;*

*var course = {*

*name: name,*

*price: price*

*};*

* Có thể viết đơn giản như sau, Sử dụng để định nghĩa key: value cho object

*var name = 'JavaScript'*

*var price = 1000;*

*var course = {*

*name,*

*price,*

*};*

* Sử dụng để định nghĩa method cho object

*var name = 'JavaScript'*

*var price = 1000;*

*var course = {*

*name,*

*price,*

*getName() {*

*return name;*

*}*

*};*

* Sử dụng để định nghĩa key cho object dưới dạng biến, biến đặt trong ngoặc vuông. Khi in ra tên mới sẽ là ReactJS chứ không còn là JavaScript nữa

*var name = 'JavaScript'*

*var price = 1000;*

*var course = {*

*[name]: 'ReactJS',*

*price,*

*};*

## Rest paramater trong ES6

Rest là lấy ra những phần còn lại

* Ví dụ sau sẽ in ra 1,2,3,4,5 vì nó sẽ lấy params và params đang là toàn bộ số truyền vào

*function logger(...params){*

*console.log(params);*

*}*

*logger(1,2,3,4,5)*

* Tuy nhiên nếu sử dụng một biến khác như sau, thì params sẽ loại bỏ phần tử đầu tiên cho biến đó, phần còn lại sẽ là params: 2,3,4,5

*function logger(a, ...params){*

*console.log(params);*

*}*

*logger(1,2,3,4,5)*

## Spread trong ES6

* Ví dụ: nối 2 mảng, spread sẽ bỏ đi dấu [] đối với mảng và {} đối với obj. Nếu trùng thuộc tính sẽ lấy cái sau

*var arr1 = [1, 2]*

*var arr2 = [3, 4, 5]*

*var arr3 = [...arr1, ...arr2]*

*console.log(arr3);*

## Detructuring trong ES6

Phân rã cấu trúc

* Ví dụ này sẽ ra 2 biến a và c, là 1 và 3. Số 2 sẽ không được định nghĩa, bỏ qua bằng cách phẩy luôn và không ghi gì

*var arr = [1, 2, 3]*

*var [a, , c] = arr*

*console.log(a, c);*

* Ví dụ 2: Khi gán biến phải trùng với thuộc tính của object, có thể đổi bằng dấu “:”. Nếu đặt thuộc tính không có trong object, có thể cho nó một giá trị mặc định nếu thuộc tính đó không tồn tại, nếu tồn tại sẽ lấy giá trị đã tồn tại đó

*var course = {*

*name: 'Javascript',*

*price: 1000,*

*children: {*

*name: 'ReactJS'*

*}*

*};*

*var {name: parenName, price, children: {name}, description = ‘default’} = course*

*console.log(parenName, price, name) // Javascript 1000 ReactJS*

# TỔNG QUAN VỀ REACT

## Khái niệm React

React là một thư viện JavaScript phổ biến để xây dựng các giao diện người dùng dựa trên thành phần, có thể tái sử dụng cho các trang web hoặc ứng dụng.

React kết hợp HTML với chức năng JavaScript thành ngôn ngữ đánh dấu của riêng nó được gọi là JSX. React cũng giúp dễ dàng quản lý luồng dữ liệu xuyên suốt ứng dụng.

React là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook để xây dựng giao diện người dùng cho các ứng dụng web đơn trang và đa trang. Nó cho phép phát triển các thành phần UI động, có thể tái sử dụng và dễ bảo trì trong các ứng dụng web lớn và phức tạp. React sử dụng mô hình hướng thành phần, nghĩa là các thành phần UI được xây dựng bằng cách kết hợp các thành phần nhỏ hơn. React cũng được sử dụng rộng rãi để phát triển các ứng dụng di động bằng React Native.

Hiện tại chỉ có 2 phiên bản chính của React là:

* ReactJS: là phiên bản chính của React, được sử dụng để phát triển các ứng dụng web đơn trang và đa trang.
* React Native: là một phiên bản của React được sử dụng để phát triển các ứng dụng di động cho iOS và Android.
* React VR: (không phổ biến) dành cho phát triển các ứng dụng thực tế ảo và React 360 cho phát triển các ứng dụng thực tế ảo trên web.

## SPA/MPA

SPA là Single-Page Application: Ứng dụng một trang: Facebook, Google, Twitter, … SPA tiếp cận hiện đại hơn, không tải lại trang trong quá trình sử dụng

MPA là Multi-Page Application: Ứng dụng nhiều trang, là cách tiếp cận cổ điển, tải lại trang trong quá trình sử dụng

# PHÂN TÍCH CÁC THÀNH PHẦN REACT

## Tạo phần tử bằng JavaScript (DOM)

### document.createElement()

Sử dụng để tạo một element

* Tạo một thẻ h1 gán cho hằng số h1, hiện tại h1 đang là một object

*const h1 = document.createElement('h1');*

* Có thể thêm đoạn văn bản bên trong h1 bằng innerText

*h1.innerText = 'Hello';*

* Tuy nhiên h1 vẫn chưa được render ra màn hình, mới chỉ trong phần code console thôi, để render, chúng ta có thể thử hàm appendChild() để cho h1 vào con của thẻ khác

*document.body.appendChild(h1);*

Như vậy thẻ h1 đã là con của thẻ body và sẽ được hiển thị trên màn hình

* Để thêm thuộc tính cho thẻ h1 ta làm như sau, thêm id, className, style

*h1.id = 'heading';*

*h1.className = 'title title2';*

*h1.style.color = 'red';*

* Cách 2: Sử dụng Object.assgin để gán cho nhiều dữ liệu, lưu ý tên các thuộc tính sẽ khác một chút so với CSS, dấu ‘-‘ trong CSS được loại bỏ và chữ tiếp theo được viết hoa, ví dụ:

*Object.assign(h1.style, {*

*color: 'red',*

*backgroundColor: '#ccc'*

*})*

## Gọi phần tử bằng JavaScript (DOM)

### document.querySelector()

Lấy ra phần tử đầu tiên của HTML với bộ chọn CSS chỉ định, tuy nhiên chỉ lấy phần tử đầu tiên nó tìm thấy

*document.querySelector('#heading') // lấy ra id có tên là heading*

*document.querySelector('.title') // lấy ra class đầu tiên có tên là title*

## Thêm React vào website

Các bước để thêm React vào website, có thể lấy đường link trực tiếp (năm 2023) hoặc sau này sẽ có một số bản sửa đổi thì truy cập vào trang web để lấy link

**Tóm gọn:**

Truy cập <https://legacy.reactjs.org/docs/add-react-to-a-website.html>

Phần step2: add the Script Tags copy đoạn và dán lên <head></head> của html trong file của bạn:

<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>

## Tạo phần tử bằng React

### React.createElement()

* Tạo một thẻ h1

*const h1 = React.createElement(‘h1’);*

* Thêm lớp, thêm id.

Cấu trúc trong ngoặc (type, props, children)

type: tên thẻ, ...

prop: được đặt trong ngoặc nhọn { } bao gồm các thuộc tính, nếu 1 thuộc tính thì không cần ngoặc nhọn

children: inner bên trong thẻ có thể là text, object, array, ....

const h1 = React.createElement('h1', {

title: 'Hello',

className: 'heading'

}, 'Hello world')

## React DOM

Dùng để render ra màn hình chính, cú pháp thêm phần reactDOM như sau:

<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>

<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>

Cấu trúc ReactDOM:

const container = getElementById(‘id of contaner’);

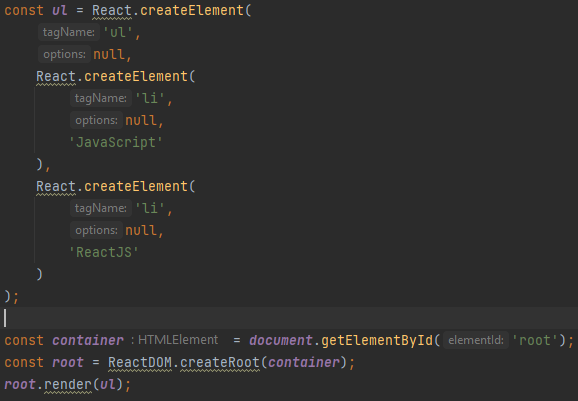
const root = ReactDOM.createRoot(container);

root.render(element);

Trong đó:

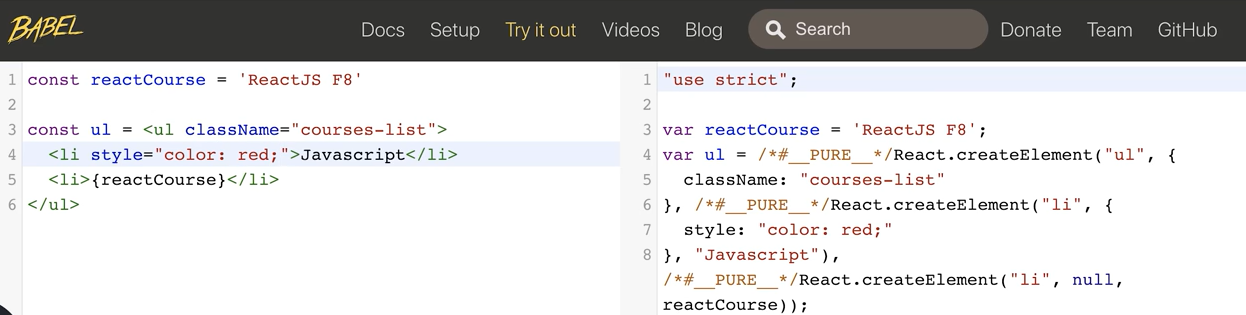
* Element: biến chứa thẻ con
* Container: biến chứa thẻ cha

VD1: Tạo thẻ 1 list gồm ReactJS và JavaScript



## JSX

JSX là JavaScript XML, XML khác với HTML nhưng về cơ bản 2 cái này khá giống nhau, nhưng để chạy được JSX cần có babel để chuyển đổi cú pháp ES6 trở thành ES5 để phù hợp cho nhiều loại trình duyệt



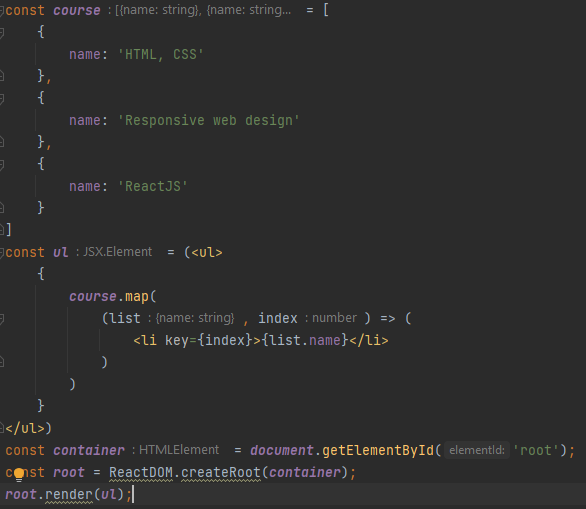
Để sử dụng babel ta khai báo như sau dưới ReactDOM

<script src="https://unpkg.com/@babel/standalone@7.15.0/babel.min.js"></script>

Và phải định dạng file script với type=”text/babel”:

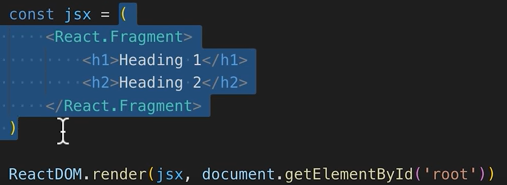
<script type="text/babel" src="./script.js"></script>

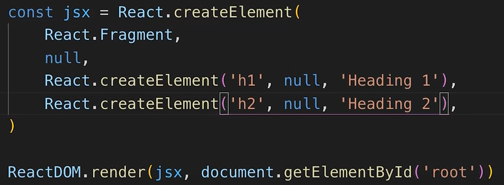
VD:



Trong JSX sẽ chỉ được tạo và render 1 thẻ trong 1 lần tạo, muốn tạo nhiều phải tạo và render nhiều, để khắc phục được điều đó thì chúng ta phải wrap chúng lại trong 1 thẻ cha, tuy nhiên nếu không muốn tạo ra một thẻ cha (ví dụ như div) thì có thể sử dụng thẻ <React.Fragment></React.Fragment> sẽ không sinh ra thẻ cha nhưng vẫn wrap được

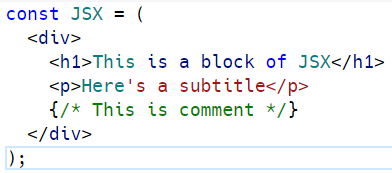
Có thể sử dụng viết tắt <></> thay cho React.Fragment trong React





Cách thêm comment trong JSX: sử dụng cặp thẻ {/\* \*/}

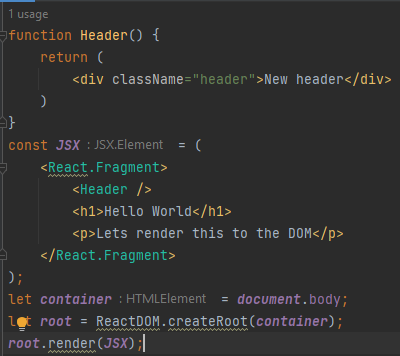
VD:



## React element types

Khi chúng ta tạo một phần tử: React.createElement(types, props, children);

Type có thể là string, function hay class. Chẳng hạn chuỗi ‘h1’ sẽ tạo ra thẻ h1



Đối với function trong React sẽ phải viết hoa biến function chữ cái đầu. Function được gọi là React Component vì có thẻ tách rời ra sử dụng nhiều nơi, còn bên trong sẽ gọi là React Element.

## Props

Các cú pháp của html vẫn sẽ được sử dụng trong React Element nhưng sẽ có một vài thay đổi, ví dụ : for trong html sẽ là forcus vào id đối tượng nhưng trong React lại là vòng lặp nên for được đổi thành htmlFor, class trong React là lớp nên khi muốn sử dụng class html phải đổi thành className.

Props là đối số của component, props có thể là bất cứ dữ liệu gì.

VD:

const ***data*** = [  
 {  
 id: 1,  
 title: 'This is title 1',  
 content: 'This is content 1'  
 },  
 {  
 id: 2,  
 title: 'This is title 2',  
 content: 'This is content 2'  
 }  
]  
const Item = function (props) {  
 return (  
 <div>  
 <h1 style={{color: 'red'}}>{props.title}</h1>  
 <h2 style={{color: 'orange'}}>{props.content}</h2>  
 </div>  
 )  
}  
function App () {  
 return (  
 <div id="wrapper">  
 {  
 ***data***.map(  
 (list, index) => (  
 <Item  
 key={index}  
 title = {list.title}  
 content = {list.content}  
 />  
 )  
 )  
 }  
 </div>  
 )  
}  
const ***container*** = ***document***.getElementById('root');  
const ***root*** = ReactDOM.createRoot(***container***);  
***root***.render(<App />);

## DOM events

DOM events là các sự kiện xảy ra khi chúng ta tương tác với đối tượng, ví dụ click, change, copy, ...

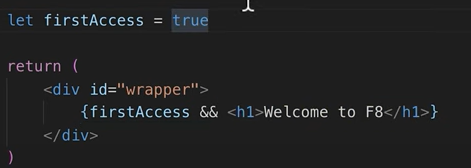
Ví dụ: với sự kiện onClick khi nhấn vào sẽ chạy cảnh báo, quy ước đặt biến sự kiện là dùng handle + tên sự kiện.

const ***data*** = [  
 {  
 id: 1,  
 title: 'This is title 1',  
 content: 'This is content 1'  
 },  
 {  
 id: 2,  
 title: 'This is title 2',  
 content: 'This is content 2'  
 }  
]  
const Item = function (props) {  
 const handleClick = () => {  
 alert(props.title);  
 }  
 return (  
 <div>  
 <h1 style={{color: 'red'}}>{props.title}</h1>  
 <h2 style={{color: 'orange'}}>{props.content}</h2>  
 <button onClick={handleClick}>Click me</button>  
 </div>  
 )  
}  
function App () {  
 return (  
 <div id="wrapper">  
 {  
 ***data***.map(  
 (list, index) => (  
 <Item  
 key={index}  
 title = {list.title}  
 content = {list.content}  
 />  
 )  
 )  
 }  
 </div>  
 )  
}  
const ***container*** = ***document***.getElementById('root');  
const ***root*** = ReactDOM.createRoot(***container***);  
***root***.render(<App />);

## Component linh hoạt

Trong JSX sẽ không in ra kiểu dữ liệu boolean, undefined, null. Có thể sử dụng điều này để kiểm tra

Ví dụ: Nếu là true thì hiển thị h1, còn nếu false thì không hiển thị



Ví dụ 1: Sử dụng type để tùy chọn loại form

// UI component  
const ***From*** = {  
 Input () {  
 return <input type="text" />  
 },  
 Checkbox () {  
 return <input type="checkbox" />  
 }  
}  
// Container  
function App () {  
 const type = 'Input';  
 const Conponent = ***From***[type];  
 return (  
 <div id="wrapper">  
 <Conponent />  
 </div>  
 )  
}  
const ***container*** = ***document***.getElementById('root');  
const ***root*** = ReactDOM.createRoot(***container***);  
***root***.render(<App />);

Ví dụ 2: Đây là cách viết đoạn mã thông thường để in ra một nút Click me với biến title là tự đặt

function Button({title}) {  
 return <button>{title}</button>  
}  
function App () {  
 return (  
 <div id="wrapper">  
 <Button title="Click me"></Button>  
 </div>  
 )  
}  
const ***container*** = ***document***.getElementById('root');  
const ***root*** = ReactDOM.createRoot(***container***);  
***root***.render(<App />);

Ví dụ 3: Trong đoạn code sau, việc sử dụng biến children là biến đặc biệt của React, và nó sẽ tự hiểu nội dung bên trong thẻ Button sẽ là children nên không cần gán giá trị nó vẫn được hiểu và không cần để trong dấu ngoặc kép, mặc định là string literals

function Button({children}) {  
 return <button>{children}</button>  
}  
function App () {  
 return (  
 <div id="wrapper">  
 <Button>Click me</Button>  
 </div>  
 )  
}  
const ***container*** = ***document***.getElementById('root');  
const ***root*** = ReactDOM.createRoot(***container***);  
***root***.render(<App />);

## NodeJS

NodeJS là JavaScript runtime là một môi trường độc lập để chạy JavaScript, thông thường ta sử dụng trình duyệt để chạy JavaScript. NodeJS không liên quan tới trình duyệt hoàn toàn chạy độc lập.

Sử dụng NodeJS dùng cho cả font-end cả back-end. Trong bài React này NodeJS sẽ được sử dụng để làm máy chủ

## Webpack

Trong React App sẽ có tích hợp webpack rồi nên sẽ không cần cài webpack, phần này sẽ nói về lý thuyết webpack.

Webpack giúp module hóa font-end

## React webpack

* Bước 1: tạo thư mục chứa project
* Bước 2: bật terminal và trỏ và thư mục chứa project (vd: react-webpack), gõ npm init
* Bước 3: gõ enter liên tiếp để cài mọi thứ mặc định, sau đó chúng ta sẽ có file package.json
* Bước 3: tại terminal gõ:

npm install webpack webpack-cli --save-dev

* Bước 4: gõ tiếp:

npm i react@latest react-dom@latest

* Bước 5: gõ tiếp:

npm install @babel/core babel-loader @babel/preset-env @babel/preset-react --save-dev

* Bước 6: tạo file public/index.html và nhấn ! tab để tạo ra 1 template html, thêm thẻ div id root ở body
* Bước 7: tạo file src/index.js

paste đoạn đơn giản sau:

import React from 'react' // nạp thư viện react  
import ReactDOM from 'react-dom/client' // nạp thư viện react-dom  
  
// Tạo component App  
function App() {  
 return (  
 <div>  
 <h1>Xin chào</h1>  
 </div>  
 )  
}  
  
// Render component App vào #root element

const root = ReactDOM.createRoot(***document***.getElementById('root'));  
root.render(<App />);

* Bước 8: cấu hình webpack, tại terminal gõ:

npm install css-loader style-loader --save-dev

* Bước 9: đặt file webpack.config.js ở thư mục gốc, ngang hàng với package.json

const path = ***require***("path");  
  
***module***.exports = {  
 entry: "./src/index.js", // Dẫn tới file index.js ta đã tạo  
 output: {  
 path: path.join(***\_\_dirname***, "/build"), // Thư mục chứa file được build ra  
 filename: "bundle.js" // Tên file được build ra  
 },  
 module: {  
 rules: [  
 {  
 test: /\.js$/, // Sẽ sử dụng babel-loader cho những file .js  
 exclude: /node\_modules/, // Loại trừ thư mục node\_modules  
 use: ["babel-loader"]  
 },  
 {  
 test: /\.css$/, // Sử dụng style-loader, css-loader cho file .css  
 use: ["style-loader", "css-loader"]  
 }  
 ]  
 },  
 // Chứa các plugins sẽ cài đặt trong tương lai  
 plugins: [  
 ]  
};

* Bước 10: Tạo file .babelrc ở thư mục gốc

{  
 "presets": [  
 "@babel/preset-env",  
 "@babel/preset-react"  
 ]  
}

* Bước 11: Tại file package.json thêm đoạn sau vào cuối “script” (trong script nhưng ở vị trí cuối)

"start": "webpack --mode development --watch",  
"build": "webpack --mode production"

* Bước 12: Tại terminal gõ

npm start

lúc này 1 file build/bundle.js đã được tự động build. Lúc này terminal đang có dạng không thể gõ được

* Bước 13: Tại ternimal gõ Ctrl + C để thoát, gõ tiếp

npm install html-webpack-plugin --save-dev

* Bước 14: Tại file webpack.config.js thêm lệnh này trước module.exports

const HtmlWebpackPlugin = require("html-webpack-plugin");

Và thêm đoạn sau trong mảng plugin

new HtmlWebpackPlugin({  
 template: "./public/index.html"  
})

* Bước 15: tại terminal gõ

npm start

lúc này sẽ tạo ra 1 file build/index.html và dòng script trong đó, tuy nhiên nó sẽ được đặt trên head, mặc dù vậy nhưng nó vẫn tải nhưng đợi html chạy xong nó mới chạy vì có từ khóa defer, làm vậy không cần cài thủ công script trong cuối thẻ body mà vẫn chạy sau cùng

* Bước 16: Ctrl + C ternminal để thoát, gõ

npm install webpack-dev-server --save-dev

* Bước 17: Sửa file webpack.config.js

Xóa dòng: "start": "webpack --mode development --watch",

Thay bằng: "start": "webpack-dev-server --mode development --open --hot",

* Bước 18: Tại terminal gõ

npm start để chạy chương trình

Cập nhật phiên bản react và react-dom mới nhất: npm i react@latest react-dom@latest

Chuyển về phiên bản cụ thể (vd 17.0.2): npm i [react@17.0.2](mailto:react@17.0.2) [react-dom@17.0.2](mailto:react-dom@17.0.2)

## Create React app

Sẽ sử dụng một câu lệnh duy nhất thay cho 18 bước phía trên:

npx create-react-app tên dự án

## NPM, NPX, YARN

NPM, NPX, YARN đều là công cụ quản lý gói phát triển JavaScript, nhưng có một số khác biệt về hiệu suất và tính năng. NPM là công cụ mặc định đi kèm với Node.js, trong khi NPX giúp thực thi các lệnh từ các gói npm mà không cần cài đặt chúng. Yarn là một công cụ thay thế cho NPM, với hiệu suất tốt hơn và các tính năng bổ sung.

* NPM (Node Package Manager): NPM là một công cụ quản lý gói mặc định cho Node.js. Nó được cài đặt cùng với Node.js và được sử dụng để tìm, cài đặt, và quản lý các gói JavaScript từ kho lưu trữ gói npm. NPM cho phép bạn quản lý các phụ thuộc (dependencies) của ứng dụng và giúp xây dựng quy trình xây dựng (build process) đơn giản hơn.
* NPX: NPX là một công cụ đi kèm với NPM từ phiên bản 5.2.0 trở đi. NPX được sử dụng để thực thi các lệnh từ các gói npm mà không cần cài đặt chúng một cách toàn cục. Nó giúp bạn chạy các tệp lệnh (executable) từ các gói npm cục bộ hoặc từ các gói npm trên kho lưu trữ mà không cần cài đặt chúng trước.
* Yarn: Yarn là một công cụ quản lý gói JavaScript phổ biến khác, được phát triển bởi Facebook và các công ty khác. Yarn có thể thực hiện các chức năng tương tự như NPM, nhưng thường được cho là nhanh hơn và có khả năng quản lý phụ thuộc tốt hơn. Yarn sử dụng một file lock để đảm bảo tính nhất quán của các phiên bản gói và hỗ trợ các tính năng như parallel installation (cài đặt song song) và caching (lưu trữ cache) để tăng tốc quá trình cài đặt.

Sự khác biệt giữa yarn và npm:

* npm: cài thư viện tuần tự, nên dùng cho ít bộ nhớ
* yarn: cài thư viện song song, nhanh hơn cho nhiều bộ nhớ

## Hooks

Hooks là những method, hàm được viết sẵn và cung cấp bởi ReactJS, mỗi hàm có 1 tính năng, 1 trường hợp cụ thể để sử dụng. Tác dụng để gắn vào các component.

Hooks chỉ dùng cho function component, không dùng được cho class component

### useState

Khi muốn dữ liệu thay đổi thì giao diện tự động cập nhật (render lại theo dữ liệu).

* Component được re-render sau khi được setState
* Initial state chỉ dùng cho lần đầu
* Set state với callback
* Initial state với callback
* Set state là thay thế state bằng giá trị mới

VD: Khi click vào nút increase thì sẽ tăng 1 giá trị

import {useState} from "react";  
function App() {  
 const [counter, setCounter] = useState(1);  
 const handleIncrease = () => {  
 setCounter(counter + 1)  
 }  
 return (  
 <div className="App" style={{padding: 20}}>  
 <h1>{counter}</h1>  
 <button onClick={handleIncrease}>Increase</button>  
 </div>  
 );  
}  
  
export default App;

Các bước thực thi:

* Bắt đầu từ App()
* Khởi tạo counter = 1 với lần khởi tạo đầu tiên, với những lần render sau sẽ không được gán vì initial state chỉ dùng cho lần đầu
* Tạo hàm handleIncrease nhưng chưa chạy gì bên trong
* Render giao diện gồm 1 thẻ h1 và 1 nút increase
* Sau khi nhấn nút increse sẽ bắt đầu chạy handleIncreate và setCounter cho counter và chỉ khi kết thúc handleIncrease mới cập nhật giá trị state và render lại

Chú ý: Chỉ khi kết thúc function đó giá trị state mới được cập nhật do đó mọi lần gọi state bên trong hàm sẽ chỉ lấy giá trị cũ. Ví dụ:

const handleIncrease = () => {  
 setCounter(counter + 1);  
 setCounter(counter + 1);  
}

Đoạn mã sau dù có 2 lần gán nhưng vẫn chỉ lấy giá trị cũ của state là 1 để gán, vậy nên kết quả 2 lần này đều là 2 chứ không phải 3

Như đã biết state chỉ cập nhật khi chạy xong function, tuy nhiên với callback set state sẽ lấy giá trị mới nhất để thực hiện sau khi kết thúc function mới cập nhật giá trị state. Ví dụ:

const handleIncrease = () => {  
 setCounter(prevState => prevState + 1);  
 setCounter(prevState => prevState + 1);  
}

Hàm này, dòng 1 sẽ cho state tạm thời lên 2, dòng 2 sẽ cho state lên 3 và sau khi kết thúc fucntion thì state lúc này đang là 1 đổi thành 3.

Nếu như giá trị khởi tạo được lấy từ một phép tính toán nào đó, chẳng hạn như 1 phép tính mất nhiều thời gian, tính toán xong rồi gán cho initial điều này làm cho việc re-render lại phải tính toán lại rồi lại gán cho initial trong khi initial chỉ được sử dụng trong lần render đầu tiên, việc lại gây mất thời gian thừa thãi, vì vậy nếu được hay đưa cả phép tính đó vào initial và sử dụng callback rồi return kết quả. Ví dụ:

Không nên:

const arr = [1, 2, 3, 4, 5];  
  
function App() {  
 const sum = arr.reduce((a, b) => a + b, 0);  
 const [counter, setCounter] = useState(sum);  
 const handleIncrease = () => {  
 setCounter(counter + 1);  
 }  
  
 return (  
 <div className="App" style={{padding: 20}}>  
 <h1>{counter}</h1>  
 <button onClick={handleIncrease}>Increase</button>  
 </div>  
 );  
}

Nên:

const arr = [1, 2, 3, 4, 5];  
function App() {  
 const [counter, setCounter] = useState(  
 arr.reduce((a, b) => a + b, 0)  
 );  
 const handleIncrease = () => {  
 setCounter(counter + 1);  
 }  
  
 return (  
 <div className="App" style={{padding: 20}}>  
 <h1>{counter}</h1>  
 <button onClick={handleIncrease}>Increase</button>  
 </div>  
 );  
}

#### One-way, Two-way binding trong React

One-way binding (kết nối một chiều) là một mô hình liên kết dữ liệu trong ReactJS, trong đó dữ liệu chỉ được truyền từ component cha (hoặc component gốc) xuống các component con thông qua các props. Trong one-way binding, các component con không thể thay đổi giá trị của props, chỉ có thể sử dụng giá trị đó để hiển thị hoặc thực hiện các hành động khác. Tóm lại kết nối một chiều chỉ làm từ 1 phía và phía kia không làm ngược lại được.

* Ví dụ 1: Trong ví dụ này mỗi lần nhập chữ trên input thì setName sẽ được gán giá trị đó, như này là kiểu kết nối 1 chiều, từ giao diện làm thay đổi code

import {useState} from "react";  
  
function App() {  
 const [name, setName] = useState('');  
 ***console***.log(name);  
 return (  
 <div>  
 <input type="text"  
 onChange={a => setName(a.target.value)}  
 />  
 </div>  
 );  
}  
  
export default App;

* Hãy thử kết nối ngược lại như sau, cho một nút Change và khi click vào thì setName sẽ được đặt mới theo code back-end quy ước, tuy nhiên nó mới chỉ hiện thị ra console chưa thực sự render ra giao diện.

<div>  
 <input type="text"  
 onChange={a => setName(a.target.value)}  
 />  
 <button onClick={() => setName('Nguyen Van A')}>Change</button>  
</div>

* Giờ hãy thử truyền lên giao diện, như vậy đoạn code này đã trở thành kết nối 2 chiều, từ UI kết nối được backend và từ backend kết nối được UI, cả 2 thay đổi qua lại lẫn nhau.

<div>  
 <input type="text"  
 value={name}  
 onChange={a => setName(a.target.value)}  
 />  
 <button onClick={() => setName('Nguyen Van A')}>Change</button>  
</div>

* Ví dụ 2: Radio button

const courses = [  
 {  
 id: 1,  
 name: 'HTML, CSS'  
 },  
 {  
 id: 2,  
 name: 'JavaScript'  
 },  
 {  
 id: 3,  
 name: 'ReactJS'  
 }  
]  
  
function App() {  
 const [checked, setChecked] = useState(1);  
 const handleSubmit = () => {  
 ***console***.log('id: ' + checked);  
 }  
 return (  
 <div>  
 {  
 courses.map((course) => (  
 <div key={course.id}>  
 <input  
 type="radio"  
 checked={checked === course.id}  
 onChange={() => setChecked(course.id)}  
 />  
 <label htmlFor="">{course.name}</label>  
 </div>  
 )  
 )  
 }  
 <button onClick={handleSubmit}>Register</button>  
 </div>  
 );  
}

Giải thích code:

Render ra 3 radio button với 3 giá trị trong mảng courses, với useState initial mặc định là chọn 1.

Trong input sẽ có thuộc tính checked nhận giá trị true hoặc false, ta đặt điều kiện cho nó, nếu như onchange là khi click vào radio nào thì sẽ setChecked cho nó là id của course đó do đó checked sẽ có giá trị là id

So sánh điều kiện checked với course.id đó nếu đúng trả về true, nếu sai trả về false, nên sẽ chi có 1 radio được chọn

Khi nhấn nút Register sẽ gọi hàm handleSubmit và sẽ log ra được id của course nào sẽ được chọn

* Ví dụ 3: Checkbox, cho phép chọn nhiều giá trị, nếu click lại sẽ hủy check

const courses = [  
 {  
 id: 1,  
 name: 'HTML, CSS'  
 },  
 {  
 id: 2,  
 name: 'JavaScript'  
 },  
 {  
 id: 3,  
 name: 'ReactJS'  
 }  
]  
  
function App() {  
 const [checked, setChecked] = useState([]);  
 const handleSubmit = () => {  
 ***console***.log({ids: checked});  
 }  
 const handleCheck = (id) => {  
 const isChecked = checked.includes(id);  
 if(isChecked){  
 setChecked(prevState => prevState.filter((x) => x!==id))  
 } else {  
 setChecked(prevState => [...prevState, id]);  
 }  
 }  
 return (  
 <div>  
 {  
 courses.map((course) => (  
 <div key={course.id}>  
 <input  
 type="checkbox"  
 checked={checked.includes(course.id)}  
 onChange={() => handleCheck(course.id)}  
 />  
 <label htmlFor="">{course.name}</label>  
 </div>  
 )  
 )  
 }  
 <button onClick={handleSubmit}>Register</button>  
 </div>  
 );  
}

Giải thích code:

Giá trị khởi tạo sẽ không còn là 1 giá trị nữa mà thay vào đó là một mảng để lưu nhiều giá trị vì checkbox cho phép chọn nhiều giá trị.

Khi nhấn nút check sẽ có sự kiện onChange và gọi ra hàm handleCheck, hàm này có tác dụng kiểm tra xem mảng checked có phần tử nào trùng với nút được click không, nếu không trùng thì thêm phần tử id mới vào mảng và lúc này chạy đến checked và checked sẽ kiểm tra được là có phần tử id đó trong mảng rồi trả về true hiển thị ô đã check. Nếu trùng thì lọc mảng và trả về mảng mới thỏa mãn điều kiện mảng không chứa id được cấp.

Diễn dải ngắn gọn hơn:

Khi muốn check => nhấn ô check => Kiểm tra có id chưa(checked-include) => Chưa=> Hiển thị chưa check => Kiểm tra id có chưa (callback) => Chưa => Thêm id mới vào mảng => Kiểm tra có id không(checked-include) => Có => Hiển thị ô đã check.

Khi muốn uncheck => nhấn ô check => Kiểm tra có id chưa(include) => Có => Hiển thị đã check => Kiểm tra id có chưa (callback) => Có => Xóa id khỏi mảng (filter) => Kiểm tra id mới có không (checked-include) => Không => Xóa bỏ ô đã check.

#### Local Storage

Là bộ nhớ trên thiết bị máy tính, cho phép lưu dữ liệu vào. Và chỉ được lưu dưới dạng chuỗi bao gồm 1 khóa và 1 string.

Cách chuyển từ mảng qua Json và từ Json sang mảng:

* const json = JSON.stringtify(tên mảng);
* const arr = JSON.parse(json);

Thêm localStorage mới:

* localStorage.setItem(‘tên khóa’, giá trị);

Lấy ra:

* const storage = localStorage.getItem(‘tên khóa’);
* const storage = localStorage.tên khóa;

Ví dụ: Làm một Todo list: Nhập các việc cần lảm rồi nhấn nút Add để hiển thị ra các list công việc cần làm, và lưu chúng vào local storage, rồi lấy từ local storage ra sử dụng.

import {useState} from "react";  
  
function App() {  
 const arr = ***JSON***.parse(***localStorage***.getItem('jobs'));  
 const [job, setJob] = useState('');  
 const [jobs, setJobs] = useState(arr ?? []);  
 const json = ***JSON***.stringify(jobs);  
 ***localStorage***.setItem('jobs', json);  
 const handleAdd = () => {  
 setJobs(prevState => [...prevState, job]);  
 setJob('');  
 }  
 return (  
 <div>  
 <input  
 value={job}  
 type="text"  
 onChange={event => setJob(event.target.value)}  
 />  
 <button onClick={handleAdd}>Add</button>  
 <ul>  
 {  
 jobs.map((element, index) => {  
 return(  
 <li key={index}>{element}</li>  
 )  
 })  
 }  
 </ul>  
 </div>  
 );  
}  
  
export default App;

Giải thích code:

* Cần 2 useState vì một cái dùng để viết (string) và một cái dùng để lưu trữ (array).
* Về phần cách hoạt động của useState xem lại mục Hooks
* Đoạn useState(arr ?? []) là giá trị khởi tạo của state init tuy nhiên toán tử “??” ở đây sẽ xác định arr có phải là null hoặc undifined hay không nếu đúng thì nó sẽ lấy [] còn không thì arr sẽ được lấy làm giá trị khởi tạo cho state

### useEffect

Thuộc dạng Hooks, sử dụng khi muốn thực hiện các Side effects. Chẳng hạn như: Update DOM, Call API, DOM event, Clean up.

Sử dụng cho các hành động yêu cầu render trước rồi mới chạy useEffect sau, tránh việc chạy trước các yêu cầu xử lý gây lỗi render.

Cấu trúc: useEffect(callback, dependence). Callback bắt buộc, dependence không bắt buộc

#### Mounted / Unmounted

Mounted là việc gắn component (file js khác) vào một file js để render, dù render hay không thì nó được gọi đến đã gọi là mounted rồi.

Unmounted là tháo ra, khi xử lý câu lệnh và render lại bị xóa đi component được mounted này thì lúc này re-render không còn xuất hiện component này nữa thì gọi là unmounted.

Callback luôn được gọi sau Mount (xem phần dưới) (phần trong useEffect luôn gọi sau quá trình render)

* useEffect(callback)

Gọi callback mỗi khi re-render, điều này gây tốn tài nguyên

Mặc dù được đọc trước khi render, nhưng nó sẽ đợi render xong nó mới chạy

VD:

import {useState, useEffect} from "react";  
function Content() {  
 const [title, setTitle] = useState('');  
 useEffect(() => {  
 ***document***.title = title;  
 })  
 return(  
 <div>  
 <input  
 type="text"  
 value={title}  
 onChange={event => setTitle(event.target.value)}  
 />  
 </div>  
 )  
}  
  
export default Content;

Giải thích:

Trong ví dụ này giải quyết việc update liên tục title của website, có thể sử dụng bên ngoài mà không cần dùng useEffect, tuy nhiên làm vậy không đúng về mặt nguyên tắc.

Hơn nữa, useEffect sẽ đợi return trên render xong nó mới chạy mặc dù theo tuần tự useEffect vẫn được đọc trước nhưng sẽ chờ để thực hiện cuối, điều này giúp giảm thiểu rủi ro khi render lỗi.

* useEffect(callback, [])

Chỉ gọi callback 1 lần sau khi component mounted

useEffect(() => {  
 ***document***.title = title;  
}, [])

* useEffect(callback, [dependence])

Gọi callback 1 lần sau khi component mounted, tuy nhiên khi dependence(deps) thay đổi (so sánh ===) thì sẽ gọi callback lần nữa còn nếu không thì sẽ không callback nữa.

import {useState, useEffect} from "react";  
function Content() {  
 const [title, setTitle] = useState('');  
 const [post, setPosts] = useState([]);  
  
 const tabs = ['posts', 'comments', 'albums', 'photos', 'todos', 'users'];  
 const [type, setType] = useState('posts');  
 useEffect(() => {  
 fetch(`https://jsonplaceholder.typicode.com/${type}`)  
 .then(res => res.json())  
 .then(posts => {  
 setPosts(posts);  
 })  
 }, [type])  
  
 return(  
 <div>  
 {tabs.map((tab, index) => (  
 <button key={index} style={(type === tab) ? {color: '#fff', backgroundColor: '#000'} : {}} onClick={() => {setType(tab)}}>{tab}</button>  
 ))}  
 <input  
 type="text"  
 value={title}  
 onChange={event => setTitle(event.target.value)}  
 />  
 <ul>  
 {  
 post.map((post) => (  
 <li key={post.id}>{post.title || post.name}</li>  
 ))  
 }  
 </ul>  
 </div>  
 )  
}  
  
export default Content;

#### UseEffect cùng với DOM event

Bảng danh sách các DOM event xem lại [link](https://www.w3schools.com/jsref/dom_obj_event.asp)

Các DOM event hay dùng: click, dblclick, focus, submit, toggle, change, scroll, resize

* Listen DOM event

useEffect( () => {  
 const handleScroll = () => {  
 setShowGoToTop(***window***.scrollY >= 200)  
 }  
 ***window***.addEventListener('scroll', handleScroll)  
}, [])  
  
return(  
 <div>  
 { showGoToTop && <button style={{position: 'fixed', bottom: '2rem', right: '2rem'}}>Go to top</button>}  
 </div>

* Những vấn đề xảy ra khi listen DOM event

Tuy nhiên, khác với các component khác, window sẽ chạy ngay cả khi unmount, và chỉ tắt khi đóng cửa sổ trình duyệt. Điều này gây rò rỉ bộ nhớ

* Cách khắc phục (clean up)

Sẽ sử dụng hàm dọn dẹp removeEventListener trước khi unmount. Các trường hợp sử dụng hàm dọn dẹp như: setInterval, setTimeout, async, EvenListener, subcribeEvent, ...

Clean up sẽ được gọi trước khi unmount, tuy nhiên với những lần callback lần 2 trở đi thì nó sẽ chạy trước khi thực hiện (lần 1 không chạy clean up – dành cho đầu lần 2).

* Dọn dẹp trước khi chạy lần 2, lần 1 không dọn.

VD:

useEffect(() => {  
 ***console***.log('mounted or re-render')  
 return () => {  
 ***console***.log('clean up')  
 }  
}, [countdown])

lần 1: mounted or re-render

lần 2: clean up – mounted or re-render

lần 3: clean up – mounted or re-render

useEffect( () => {  
 const handleScroll = () => {  
 setShowGoToTop(***window***.scrollY >= 200)  
 }  
 ***window***.addEventListener('scroll', handleScroll)  
   
 return () => {  
 ***window***.removeEventListener('scroll', handleScroll)  
 }  
}, [])

#### SetInterval, Setimeout

Cả 2 đều chỉ thực thi mã trong khoảng thời gian nhất định, có sử dụng clean up trước khi unmount.

SetInterval sẽ thực hiện lặp đi lặp lại trong 1 khoảng thời gian cố định

SetTimeout sẽ thực hiện sau 1 khoảng thời gian nhất định

import {useState, useEffect} from "react";  
  
function Content() {  
 const [countdown, setCountdown] = useState(180)  
  
 useEffect(() => {  
 const timerID = setInterval(() => {  
 setCountdown(prevState => prevState - 1)  
 }, 1000)  
 return () => {  
 clearInterval(timerID)  
 }  
 }, [countdown])  
 return (  
 <div>  
 <h1>{countdown}</h1>  
 </div>  
 )  
}  
  
export default Content;

VD2: Chọn 1 ảnh và hiển thị nó lên màn hình:

* Có sự re-render nên dùng useState
* Khi thay đổi ảnh lần 2 trở đi thì vẫn thao tác trên bộ nhớ đó nên vẫn phải có clean up
* Dùng clean up thì phải dùng useEffect

import {useState, useEffect} from "react";  
  
function Content() {  
 const [avatar, setAvatar] = useState();  
  
 useEffect(() => {  
 return () => {  
 avatar && ***URL***.*revokeObjectURL*(avatar.preview);  
 }  
 }, [avatar])  
 const handlePreviewAvatar = (e) => {  
 const file = e.target.files[0];  
 file.preview = ***URL***.*createObjectURL*(file);  
 setAvatar(file);  
 }  
 return (  
 <div>  
 <input type="file" onChange={handlePreviewAvatar}/>  
 { avatar && <img src={avatar.preview} alt=""/>}  
 </div>  
 )  
}  
  
export default Content;

#### Subcribe, Unsubcribe

Tạo ra event sử dụng global toàn dự án: window.dispatchEvent()

VD: Tạo ra một list các bài học, khi nhấn vào bài nào sẽ hiện những cmt của bài đó 2 giây 1 lần làm mới.

* Tạo event toàn cục sử dụng tất cả các file đều được: window.dispatchEvent(), lưu tại index.js

import React from 'react';  
import ReactDOM from 'react-dom/client';  
import './index.css';  
import App from './App';  
import reportWebVitals from './reportWebVitals';  
  
function emitComment(id) {  
 setInterval(() => {  
 ***window***.dispatchEvent(  
 new CustomEvent(`lesson-${id}`,{  
 detail: `Noi dung comment cua lesson ${id}`  
 })  
 )  
 }, 2000)  
}  
emitComment(1)  
emitComment(2)  
emitComment(3)  
  
const root = ReactDOM.createRoot(***document***.getElementById('root'));  
root.render(  
 <React.StrictMode>  
 <App />  
 </React.StrictMode>  
);  
  
// If you want to start measuring performance in your app, pass a function  
// to log results (for example: reportWebVitals(console.log))  
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals  
reportWebVitals();

* Tại file làm việc Content.js hoặc App.js

import {useState, useEffect} from "react";  
  
const lessons = [  
 {  
 id: 1,  
 name: 'ReactJS'  
 },  
 {  
 id: 2,  
 name: 'SPA, MPA'  
 },  
 {  
 id: 3,  
 name: 'Arrow function'  
 }  
]  
  
function Content() {  
 const [lessonId, setLessonId] = useState(1);  
 useEffect(() => {  
 const handleComment = ({detail}) => {  
 ***console***.log(detail);  
 }  
 ***window***.addEventListener(`lesson-${lessonId}`, handleComment)  
 return () => {  
 ***window***.removeEventListener(`lesson-${lessonId}`, handleComment)  
 }  
 },[lessonId])  
 return (  
 <div>  
 <ul>  
 {lessons.map((lesson) => {  
 return (  
 <li  
 key={lesson.id}  
 style={ lessonId === lesson.id ? {color: 'red'} : {} }  
 onClick={() => {setLessonId(lesson.id)}}  
 >{lesson.name}</li>  
 )  
 })}  
 </ul>  
 </div>  
 )  
}  
  
export default Content;

### useLayoutEffect

Về cơ bản thì giống với useEffect, tuy nhiên nó thực hiện callback trước khi render mà useEffect thì thực hiện callback sau quá trình render.

Ví dụ: Khi click vào run thì giá trị tăng lên 1, nếu quá 3 trở về 0

* Trong ví dụ này nếu sử dụng useEffect sẽ có nhược điểm là khi tới số 4 nó sẽ render ra số 4 rồi mới thực hiện callback kiểm tra quá 3 không, rồi mới render về 0.

import {useState, useEffect, useLayoutEffect} from "react";  
  
function Content() {  
 const [count, setCount] = useState(0);  
 useEffect(() => {  
 if(count > 3){  
 setCount(0);  
 }  
 }, [count])  
 const handleCount = () => {  
 setCount(count + 1);  
 }  
 return (  
 <div>  
 <h1>{count}</h1>  
 <button onClick={handleCount}>run</button>  
 </div>  
 )  
}  
  
export default Content;

* Còn khi sử dụng useLayoutEffect thì nó sẽ chạy callback trước khi thực hiện render

import {useState, useEffect, useLayoutEffect} from "react";  
  
function Content() {  
 const [count, setCount] = useState(0);  
 useLayoutEffect(() => {  
 if(count > 3){  
 setCount(0);  
 }  
 }, [count])  
 const handleCount = () => {  
 setCount(count + 1);  
 }  
 return (  
 <div>  
 <h1>{count}</h1>  
 <button onClick={handleCount}>run</button>  
 </div>  
 )  
}  
  
export default Content;

So sánh quá trình xử lý của useEffect và useLayoutEffect

|  |  |
| --- | --- |
| useEffect | useLayoutEffect |
| Cập nhật lại state | Cập nhật lại state |
| Cập nhật lại DOM (mutated – đột biến, chỉ sửa lại 1 property của object) | Cập nhật lại DOM (mutated) |
| Render lại UI | Gọi cleanup nếu deps thay đổi (sync – đồng bộ) |
| Gọi cleanup nếu deps thay đổi | Gọi useLayoutEffect callback (sync) |
| Gọi useEffect callback | Render lại UI |

### useRef

Ref = reference (tham chiếu)

Sử dụng để lưu các giá trị qua một tham chiếu bên ngoài. function component.

VD: Tạo một đồng hồ đếm ngược từ 60 giây, khi nhấn start thì bắt đầu đếm ngược còn stop thì dừng

* Ta code được bài sau, tuy nhiên việc sau mỗi giây sẽ thực hiện việc callback và re-render lại, do đó timerId sẽ thuộc vùng làm việc trước và stop gọi lại sẽ là undefined chứ không lấy được timerId trước đó.

import {useState, useEffect} from "react";  
  
function Content() {  
 const [count, setCount] = useState(60);  
  
 let timerId;  
 const handlerStart = () => {  
 timerId = setInterval(() => {  
 setCount(prevState => prevState - 1)  
 }, 1000)  
 }  
 const handleStop = () => {  
 clearInterval(timerId)  
 }  
 return (  
 <div>  
 <h1>{count}</h1>  
 <button onClick={handlerStart}>Start</button>  
 <button onClick={handleStop}>Stop</button>  
 </div>  
 )  
}  
  
export default Content;

* Để khắc phục, ta có thể để bên ngoài function Content() hoặc App(), tuy nhiên trong làm việc thực tế, ta chia nhỏ các component và mỗi component làm việc với 1 chức năng riêng, không thể để bên ngoài được do đó ta dùng useRef tham chiếu đến phiên làm việc trước để giải quyết vấn đề này.

useRef sẽ nhận một giá trị khởi tạo trong lần gọi đầu tiên và trả về 1 object và được đặt biến là current. useRef không bị ảnh hưởng sau mỗi lần re-render

const ref = useRef(99); // {current: 99}

ref.current = 88; // {current: 88}

* Trong ví dụ trên mỗi lần re-render timerId sẽ có 1 giá trị khác nhau (1s render 1 lần) do đó việc handlerStop sẽ không lấy được đúng timerId lúc trước sau khi re-render nhiều lần
* Việc sử dụng useRef sẽ lưu trữ timerId thành 1 giá trị bất biến(trừ khi gán giá trị mới bằng cách truy cập biến current
* current hiện tại là underfined, sau khi handlerStart, current được gán cho timerId mới và sau mỗi lần re-render này giá trị timerId sẽ không thay đổi, do đó handlerStop có thể truy cập được và dừng lại.

import {useState, useEffect, useRef} from "react";  
  
function Content() {  
 const [count, setCount] = useState(60);  
  
 const timerId = useRef();  
 const handlerStart = () => {  
 timerId.current = setInterval(() => {  
 setCount(prevState => prevState - 1)  
 }, 1000)  
 }  
 const handleStop = () => {  
 clearInterval(timerId.current)  
 }  
 return (  
 <div>  
 <h1>{count}</h1>  
 <button onClick={handlerStart}>Start</button>  
 <button onClick={handleStop}>Stop</button>  
 </div>  
 )  
}  
  
export default Content;

### memo

memo không phải là hook. memo là ghi nhớ, là HOC (higher order component). Khi chúng ta mount một component con nhưng chỉ render đúng 1 lần và không xử lý gì nữa, mà component cha cần xử lý nhiều lần mà mỗi lần render lại gọi đến component con, điều này gây lãng phí tài nguyên. memo sinh ra để giải quyết vấn đề này.

VD:

* memo nhận vào 1 component, sau đó check các props của component, sau mỗi lần re-render có bị thay đổi hay không, nếu có thì cho re-render. Sử dụng toán tử spread === để so sánh.

import {useState, useEffect, memo} from "react";  
  
function Content() {  
 ***console***.log('re-render')  
 return (  
 <div>  
 <h1>Hello world</h1>  
 </div>  
 )  
}  
  
export default memo(Content);

KL: khi nào component có nhiều props, độ phức tạp UI lớn

### useCallback

Giúp tránh tạo ra hàm mới không cần thiết trong function component.

Luôn sử dụng với memo.

VD: Xét ví dụ sau

App.js

import {useState} from "react";  
import ***Content*** from "./Content";  
  
function App() {  
 const [count, setCount] = useState(0);  
 const handlerIncrease = () => {  
 setCount(count + 1)  
 }  
 return (  
 <div>  
 <Content onCrease={handlerIncrease}></Content>  
 <h1>{count}</h1>  
 </div>  
 );  
}  
  
export default App;

Content.js

import {useState, useEffect, memo} from "react";  
  
function Content({onCrease}) {  
 ***console***.log('re-render')  
 return (  
 <div>  
 <h1>Hello world</h1>  
 <button onClick={onCrease}>Click me</button>  
 </div>  
 )  
}  
  
export default memo(Content);

* Trong ví dụ này, mỗi lần setCount là 1 lần re-render, do đó sẽ tạo các hàm handlerIncrease liên tục mỗi lần render
* Xem ví dụ sau:

const a = () => {} // undefined

const b = () => {} // undefined

a === b // false

a tạo 1 object mới, b tạo một object mới, a === b // false vì tham chiếu 2 object khác nhau.

* Mặc dù có memo, tuy nhiên nó sẽ so sánh === với tham chiếu cũ, nhưng mỗi lần tạo 1 hàm mới là sẽ tạo một object mới, do đó việc tham chiếu sẽ không trùng nhau mặc dù cùng đoạn code vì vậy dẫn đến việc render bên Content nhiều lần không cần thiết.

Sử dụng useCallback để xử lý việc tạo những hàm không cần thiết. useCallback phải nhận 2 đối số giống như useEffect, đối số 1: callback, đối số 2: deps

import {useState, useCallback} from "react";  
import ***Content*** from "./Content";  
  
function App() {  
 const [count, setCount] = useState(0);  
 const handlerIncrease = useCallback(() => {  
 setCount(prevState => prevState + 1)  
 }, [])  
 return (  
 <div>  
 <Content onCrease={handlerIncrease}></Content>  
 <h1>{count}</h1>  
 </div>  
 );  
}  
  
export default App;

### useMemo

Là một hook, sử dụng để tránh lặp lại một logic nào đó không cần thiết

VD: Viết một chương trình nhập tên và giá sp, in ra danh sách sp kèm giá và tính tổng sp

import {useState, useMemo, useRef} from "react";  
  
function App() {  
 const [productName, setProductName] = useState('');  
 const [productPrice, setProductPrice] = useState('');  
 const [products, setProducts] = useState([]);  
  
 const refName = useRef()  
 const handlerSubmit = () => {  
 setProducts(  
 [...products,  
 {  
 name: productName,  
 price: Number(productPrice)  
 }]  
 )  
 setProductName('');  
 setProductPrice('');  
 refName.current.focus();  
 }  
  
 const total = useMemo(() => {  
 const result = products.reduce((init, prod) => {  
 return(init + prod.price)  
 }, 0)  
 return result  
 }, [products])  
 return(  
 <div>  
 <input  
 ref={refName}  
 value={productName}  
 onChange={(e) => setProductName(e.target.value)}  
 placeholder="Nhập tên sản phẩm"  
 type="text"/>  
 <br/>  
 <input  
 value={productPrice}  
 onChange={(e) => setProductPrice(e.target.value)}  
 placeholder="Nhập giá sản phẩm"  
 type="text"/>  
 <br/>  
 Total = {total}  
 <br/>  
 <button onClick={handlerSubmit}>Add</button>  
 <ul>  
 {  
 products.map((prod, index) => {  
 return(  
 <li key={index}>{prod.name} - {prod.price}</li>  
 )  
 })  
 }  
 </ul>  
 </div>  
 );  
}  
export default App

Các bước thực hiện:

* Tạo khối nhập tên, giá, hiển thị total và danh sách
* Vì tên và giá là trường nhập nên dùng useState khởi tạo chuỗi rỗng, bắt sự kiện nhập sau đó set giá trị cho useState
* Vì hiển thị ra dạng danh sách ul li nên tạo useState mảng rỗng và hàm lưu giá trị cho onClick button thêm
* Đối với hàm submit sử dụng spead ... để lưu giá trị mới và cũ sau mỗi lần submit
* Đối với total phải sử dụng useMemo vì nó là sử dụng logic để tính toán, sau mỗi lần re-render thì phương thức này sẽ bị lặp nếu không sử dụng useMemo, vậy nên muốn chỉ sử dụng khi products thay đổi thì chúng ta phải sử dụng useMemo với deps là products
* Với total sử dụng phương thức reduce tính tổng trong mảng có chứa object

.reduce((init, prod) => {return(init + prod.price)}, 0)

Giá trị khởi tạo của tổng sẽ là 0 và cộng dần với products.price

* Đối với handler submit sau khi nhấn sẽ reset lại input, và focus vào input đầu, sử dụng useRef để tham chiếu và không thay đổi qua mỗi lần render, phương thức .focus là thuộc về DOM html chứ không phải của React.

### useReducer

useReducer là một hook của react, sử dụng để quản lý trạng thái (state) trong một function component

* Nó thay thế được useState và useState cũng có thể sử dụng thay useReducer
* useState sử dụng trong các state đơn giản (state: string, number, ...), useReducer sử dụng trong các state phức tạp (state: array, object, ...)

Các bước để thực hiện useReducer

* init state
* action
* reducer (state, action)
* dispatch

VD: Bộ đếm tăng giảm sử dụng useReducer

import {useReducer} from 'react'  
  
// Init state  
const initState = 0;  
  
// Action  
const UP\_ACTION = 'up';  
const DOWN\_ACTION = 'down';  
  
// reducer  
const reducer = (state, action) => {  
 switch (action){  
 case UP\_ACTION:  
 return state + 1  
 case DOWN\_ACTION:  
 return state - 1  
 default:  
 throw new Error('Invalid Action')  
 }  
}  
  
function App() {  
 const [count, dispatch] = useReducer(reducer, initState)  
 return(  
 <>  
 <h1>{count}</h1>  
 <button onClick={() => {dispatch(UP\_ACTION)}}>UP</button>  
 <button onClick={() => {dispatch(DOWN\_ACTION)}}>DOWN</button>  
 </>  
 )  
};  
  
export default App;

VD2: Tạo một chương trình nhập vào công việc và khi nhấn Add sẽ lưu lại và in ra danh sách các công việc cần làm

* Init state

1 Tạo biến initState khởi tạo, với 2 key để lưu giá trị nhập input (kiểu string) và lưu vào mảng in ra danh sách (kiểu array)

* Action

2 Tạo 3 action tương ứng (set, add, del)

6 Tạo 3 object nhận dữ liệu truyền đi (payload) và thêm loại (type)

* Reducer

3 Tạo reducer với 2 tham số truyền vào là state hiện tại và action

7 Tạo switch phù hợp với 3 loại (type) của 3 action

9 Lưu ý: các key trùng sẽ được lấy bởi key mới nhất dán đè. trong case phải luôn có break để kết thúc case, trong switch case phải luôn có 1 default

* Dispatch

4 Tạo useReducer với 2 tham số truyền vào là reducer và giá trị khởi tạo

5 Trong trường input không thể cài dispatch thông thường như ví dụ trên vì nó sẽ cần có tham số truyền đi là dữ liệu nhập, nên cần tạo 3 function phù hợp với 3 action

8 Thêm các dispatch set, add, del cho 3 loại cần làm

import { useReducer, useRef } from 'react'  
  
// 1. Init State  
const initState = {  
 job: '',  
 jobs: []  
}  
// 2. Action  
const SET\_JOB = 'set\_job';  
const ADD\_JOB = 'add\_job';  
const DEL\_JOB = 'del\_job';  
  
const setJob = payload => {  
 return{  
 type: SET\_JOB,  
 payload  
 }  
}  
  
const addJob = payload => {  
 return{  
 type: ADD\_JOB,  
 payload  
 }  
}  
  
const delJob = payload => {  
 return{  
 type: DEL\_JOB,  
 payload  
 }  
}  
// 3. Reducer  
const reducer = (state, action) => {  
 let newState;  
 switch (action.type){  
 case SET\_JOB:  
 newState = {  
 ...state,  
 job: action.payload  
 }  
 break  
 case ADD\_JOB:  
 newState = {  
 ...state,  
 jobs: [...state.jobs, action.payload]  
 }  
 break  
 case DEL\_JOB:  
 const newJobs = [...state.jobs]  
 newJobs.splice(action.payload, 1)  
 newState = {  
 ...state,  
 jobs: newJobs  
 }  
 break  
 default:  
 throw new Error("Invalid Action")  
 }  
 return newState;  
}  
// 4. Dispatch  
function App() {  
 const [state, dispatch] = useReducer(reducer, initState)  
 const {job, jobs} = state  
 const inputRef = useRef()  
 return(  
 <>  
 <h1>*ToDo*</h1>  
 <input  
 ref={inputRef}  
 onChange={(e) => {  
 dispatch(setJob(e.target.value))  
 }}  
 value={job}  
 placeholder="Enter Todo ..."  
 type="text"/>  
 <button onClick={() => {  
 dispatch(addJob(job))  
 dispatch(setJob(''))  
 inputRef.current.focus()  
 }}>Add</button>  
 <ul>  
 {jobs && jobs.map((job, index) => {  
 return(  
 <li key={index}>  
 {job}  
 <button onClick={() => {  
 dispatch(delJob(index))  
 }}>Xóa</button>  
 </li>  
 )  
 })}  
 </ul>  
 </>  
 )  
};  
  
export default App;

Ưu điểm của useReducer là có thể bóc tách file riêng biệt theo từng chức năng và import vào được:

constants.js // chuyên viết các biến const (export const ...)

actions.js // tạo action

reducer.js // tạo reducer

dispatch.js // tạo các dispatch tương ứng action

### useContext

* React context:

Là đơn giản hóa việc truyền dữ liệu, từ component cha xuống các component con mà không cần phải sử dụng tới props

Các bước:

+ B1: Tạo context (tạo ngữ cảnh, phạm vi để truyền) ở trong component cha

import {createContext} from ‘react’

const Variable = createContext() // object {Provider, Consumer}

+ B2: Tạo provider để truyền đi

<Variable.Provider value={}> ... </Variable.Provider>

+ B3: Tạo consumer để nhận về

import {useContext} from ‘react’

import {Variable} from ‘./App’

...

VD: App>Content>Paragraph. Sử dụng trực tiếp Paragraph không cần truyền tham số qua Content để lấy.

* App.js

Tạo createContext và Provider cho component cha

import {useState, createContext} from "react";  
import Content from './Content'  
import './App.css'  
  
export const ***ThemeContext*** = createContext()  
  
function App() {  
 const [theme, setTheme] = useState('light');  
 const toggleTheme = () => {  
 setTheme(theme === 'light' ? 'dark' : 'light');  
 }  
 return(  
 <ThemeContext.Provider value={theme}>  
 <div>  
 <button onClick={toggleTheme}>Toggle theme</button>  
 <Content></Content>  
 </div>  
 </ThemeContext.Provider>  
 )  
};  
  
export default App;

* Content.js

import Paragraph from "./Paragraph";  
  
function Content() {  
 return(  
 <div>  
 <Paragraph></Paragraph>  
 </div>  
 )  
};  
  
export default Content;

* Paragraph.js

import {useContext} from 'react';  
import {***ThemeContext***} from './App'  
  
function Paragraph() {  
 const theme = useContext(***ThemeContext***)  
 ***console***.log(theme)  
 return(  
 <div>  
 <p className={theme}>  
 Lorem Ipsum is simply dummy text   
 of the printing and typesetting industry.  
 </p>  
 </div>  
 )  
}  
  
export default Paragraph;

VD2: Như ví dụ trên nhưng phân bổ thư mục bài bản

* ThemeContext.js // tạo context, tạo provider

import {useState, createContext} from "react";  
  
// Tạo context  
export const ***ThemeContext*** = createContext()  
  
// Tạo Provider  
function ThemeProvider({children}) {  
 const [theme, setTheme] = useState('light');  
 const toggleTheme = () => {  
 setTheme(theme === 'light' ? 'dark' : 'light');  
 }  
 const value = {  
 theme,  
 toggleTheme  
 }  
 return(  
 <ThemeContext.Provider value={value}>  
 {children}  
 </ThemeContext.Provider>  
 )  
}  
  
export { ThemeProvider }

* Paragraph.js // đổi nguồn từ ThemeContext.js

import {useContext} from 'react';  
import {***ThemeContext***} from './ThemeContext'  
  
function Paragraph() {  
 const context = useContext(***ThemeContext***)  
 return(  
 <div>  
 <p className={context.theme}>  
 Lorem Ipsum is simply dummy text  
 of the printing and typesetting industry.  
 </p>  
 </div>  
 )  
}  
  
export default Paragraph;

* App.js

import Content from './Content'  
import {***ThemeContext***} from './ThemeContext'  
import './App.css'  
import { useContext } from 'react'  
  
function App() {  
 const context = useContext(***ThemeContext***)  
 return(  
 <div>  
 <button onClick={context.toggleTheme}>Toggle theme</button>  
 <Content></Content>  
 </div>  
 )  
};  
  
export default App;

* index.js // vì provider sẽ ở mục cao nhất, nên index.js lớn nhất, bao trùm bởi provider

import React from 'react';  
import ReactDOM from 'react-dom/client';  
import './index.css';  
import App from './App';  
import reportWebVitals from './reportWebVitals';  
  
import { ThemeProvider} from "./ThemeContext";  
  
const root = ReactDOM.createRoot(***document***.getElementById('root'));  
root.render(  
 <React.StrictMode>  
 <ThemeProvider>  
 <App />  
 </ThemeProvider>  
 </React.StrictMode>  
);  
  
// If you want to start measuring performance in your app, pass a function  
// to log results (for example: reportWebVitals(console.log))  
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals  
reportWebVitals();

#### Global State (Context + useReducer)

* index.js

7. Lấy Provider bao hàm cho index (cha)

9. Thay đổi đường dẫn từ 8

* App.js

10. khai báo và lấy context từ 8, lấy [state, dispatch]

14. Sửa lại 10 từ 12

* Context.js

1 Tạo thư mục store lưu trữ, tạo file context

* Provider.js

2 Tạo useReducer

3 Lấy context từ 1

5 Truyền initState và reducer từ 4 sang

6 Sử dụng reducer và initState cho useReducer

* reducer.js

4 Tạo initState và reducer (export)

17 lấy constant

* index.js

8 export default các component xung quanh

13 export \* from './hooks'

16 export 14 15

* hooks.js

11 Khai báo, lấy context từ context

12 tạo useStore lấy [state, dispatch] từ context và trả về [state, dispatch]

=> chủ yếu lấy khai báo đoạn 11

* constants.js

14 Tạo các const (export)

* actions.js

15 Tạo các action

* logger.js

1. Tạo logger, nhận reducer trả về reducer trong khi nhận thì log ra những cái khác

* store/Context.js

import { createContext } from 'react';  
  
const ***Context*** = createContext();  
  
  
export default ***Context***;

* store/Provider.js

import { useReducer } from 'react';  
import ***Context*** from './Context'  
import reducer, {***initState***} from "./reducer";  
import logger from "./loggers";  
  
function Provider ({children}) {  
 const [state, dispatch] = useReducer(logger(reducer), ***initState***)  
 return(  
 <Context.Provider value={[state, dispatch]}>  
 {children}  
 </Context.Provider>  
 )  
}  
  
export default Provider;

* store/constants.js

const ***SET\_TODO*** = 'set\_todo';  
const ***ADD\_TODO*** = 'add\_todo';  
const ***DEL\_TODO*** = 'del\_todo';  
  
export { ***SET\_TODO***, ***ADD\_TODO***, ***DEL\_TODO***}

* store/action.js

import { ***SET\_TODO***, ***ADD\_TODO***, ***DEL\_TODO*** } from './constants'  
const setTodo = payload => {  
 return{  
 type: ***SET\_TODO***,  
 payload  
 }  
}  
  
const addTodo = payload => {  
 return{  
 type: ***ADD\_TODO***,  
 payload  
 }  
}  
  
const delTodo = payload => {  
 return{  
 type: ***DEL\_TODO***,  
 payload  
 }  
}  
  
export { setTodo, addTodo, delTodo };

* store/reducer.js

import { ***SET\_TODO***, ***ADD\_TODO***, ***DEL\_TODO***} from './constants'  
import {addTodo} from "./actions";  
const ***initState*** = {  
 todos: [],  
 todoInput: '',  
}  
  
function reducer(state, action) {  
 switch (action.type) {  
 case ***SET\_TODO***:  
 return {  
 ...state,  
 todoInput: action.payload  
 }  
 case ***ADD\_TODO***:  
 return{  
 ...state,  
 todos: [...state.todos, action.payload]  
 }  
 case ***DEL\_TODO***:  
 let newTodos = [...state.todos]  
 newTodos.splice(action.payload, 1)  
 return{  
 ...state,  
 todos: newTodos  
 }  
 default:  
 throw new Error("Invalid Action")  
 }  
}  
  
export { ***initState*** }  
export default reducer;

* store/hooks.js

import { useContext } from "react";  
import ***Context*** from './Context'  
  
const useStore = () => {  
 const [state, dispatch] = useContext(***Context***);  
  
 return [state, dispatch]  
}  
  
export {useStore}

* store/index.js

export {default as StoreProvider} from './Provider';  
export {***default*** as StoreContext} from './Context';  
export \* from './hooks'  
export \* as constants from './constants'  
export \* as actions from './actions'

* App.js

import { useStore, actions } from "./store";  
import {addTodo, delTodo, setTodo} from "./store/actions";  
import { useRef } from 'react'  
  
function App() {  
 const [state, dispatch] = useStore();  
 const {todos, todoInput} = state  
 const refInput = useRef()  
  
 const handleAdd = () => {  
 dispatch(addTodo(todoInput))  
 dispatch(setTodo(""))  
 refInput.current.focus()  
 }  
 return(  
 <div>  
 <input  
 ref={refInput}  
 type="text"  
 value={todoInput}  
 placeholder="Enter todo..."  
 onChange={e => {  
 dispatch(setTodo(e.target.value))  
 }}  
 />  
 <button onClick={handleAdd}>Add</button>  
 <ul>  
 {todos && todos.map((todo, index) => {  
 return(  
 <li key={index}>  
 {todo}  
 <span style={{marginLeft: '20px', color: 'red'}} onClick={() => {  
 dispatch(delTodo(index))  
 }}>X</span>  
 </li>  
 )  
 })}  
 </ul>  
 </div>  
 )  
};  
  
export default App;

* store/loggers.js

function logger(reducer) {  
 return (preState, action) => {  
 ***console***.group(action.type)  
 ***console***.log("Prev State", preState)  
 ***console***.log("Action", action)  
  
 const nextState = reducer(preState, action);  
  
 ***console***.log("Next State", nextState)  
 ***console***.groupEnd()  
 return nextState;  
 }  
}  
  
export default logger;

### useImperativeHandle

useInperativeHandle dùng để tùy chỉnh ref của function component

Thông thường trong các thẻ DOM react cung cấp một thuộc tính ref dùng để getElement thẻ đó, tuy nhiên đối với thẻ tự tạo trong react như sau:

import Video from "./Video";  
import {useRef} from "react";  
  
function App() {  
 const videoRef = useRef()  
 return(  
 <div>  
 <Video ref={videoRef}></Video>  
 <button>Play</button>  
 <button>Pause</button>  
 </div>  
 )  
}  
  
export default App;

Thẻ <Video> không phải là thẻ DOM nên truyền ref là không thể vì không có thuộc tính ref này, vì <Video> là function component không phải là DOM element

React cung cấp 1 HOC làm việc này là forwardRef, giống với memo là sẽ bao hàm conponent con, sẽ nhận 2 đối số

import video1 from './videos/video1.mp4'  
import {forwardRef} from 'react'  
  
function Video(props, ref) {  
 return(  
 <video ref={ref} src={video1} width="50%"></video>  
 )  
}  
  
export default forwardRef(Video)

import ***Video*** from "./Video";  
import {useRef} from "react";  
  
function App() {  
 const videoRef = useRef()  
 const handlerPlay = () => {  
 videoRef.current.play();  
 }  
 const handlerPause = () => {  
 videoRef.current.pause();  
 }  
 return(  
 <div>  
 <Video ref={videoRef}></Video>  
 <button onClick={handlerPlay}>Play</button>  
 <button onClick={handlerPause}>Pause</button>  
 </div>  
 )  
}  
  
export default App;

Tuy nhiên có thể thấy được rằng bên Video.js không có tính đóng gói, vì dữ liệu bên Video.js lại truyền được sang bên App.js, vì App.js lại thao tác và lấy được element từ Video.js. Điều này gây nguy hiểm cho nhà phát triển, để khắc phục điều này, ta làm như sau:

Video.js

import video1 from './videos/video1.mp4'  
import {forwardRef, useImperativeHandle, useRef} from 'react'  
  
function Video(props, ref) {  
 const videoRef = useRef()  
 useImperativeHandle(ref, () => ({  
 play() {  
 videoRef.current.play()  
 },  
 pause () {  
 videoRef.current.pause()  
 }  
 }))  
 return(  
 <video ref={videoRef} src={video1} width="50%"></video>  
 )  
}  
  
export default forwardRef(Video)

App.js

import ***Video*** from "./Video";  
import {useRef} from "react";  
  
function App() {  
 const videoRef = useRef()  
 const handlerPlay = () => {  
 videoRef.current.play();  
 }  
 const handlerPause = () => {  
 videoRef.current.pause();  
 }  
 return(  
 <div>  
 <Video ref={videoRef}></Video>  
 <button onClick={handlerPlay}>Play</button>  
 <button onClick={handlerPause}>Pause</button>  
 </div>  
 )  
}  
  
export default App;

Giải thích: ý tưởng là chỉ đưa 2 chức năng play và pause ra cho app, còn dữ liệu thì vẫn để ở video

* Tạo function component Video và export nó ra, do là function component nên để sử dụng ref cần phải dùng HOC forwardRef bọc Video lại, khai báo bên App.js
* Lúc này Video sẽ nhận 2 tham số, tham số thứ 2 chính là tham số truyền đi
* Sử dụng useImperativeHandle để chỉ truyền phương thức sang chứ không truyền video sang
* useImterativeHandle(ref<tham số thứ 2>, function<bao gồm các phương thức>
* useImterativeHandle tham số 1 (ref) sẽ sử dụng trong component con đó, còn tham số thứ 2 là function sẽ định nghĩa những phương thức tương tác với ref, và đưa những phương thức đó cho component cha chứ không đưa ref cho component cha

## CSS

### CSS inline

Sử dụng ngay trong thẻ DOM element, không khuyến khích vì đây là code cứng, khó debug sau này

VD:

<h1 style={{color: ‘red’}}>Hello</h1>

### Sử dụng CSS external

import ‘./style.css’

Tuy nhiên nó sẽ hiển thị trên trình duyệt là dạng internal <style>...</style> và nhiều file css sẽ có nhiều thẻ style

Và thông thường khi run build (npm run build) cho production sẽ gộp lại làm 1 file css điều này gây xung đột giữa các file css, css sau sẽ gán đề lên file trc nếu trùng className

### CSS module

Là phương pháp cấu trúc hóa CSS trong ứng dụng web tránh xung đột giữa phạm vi các file CSS

Để sử dụng CSS module, cần sửa file css có dạng .module.css, Ví dụ App.module.css và nó sẽ export 1 object tên là styles, lưu ý đặt tên className theo CamelCase vì sẽ tiện cho sử dụng js, trong js không được dùng dấu -

VD: header-bottom => headerBottom

import styles from ‘./App.module.css

object style sẽ chứa các class làm key và value sẽ là 1 đoạn chuỗi cấu tạo từ nhiều phần

VD:

.module.css:

.paragraph{  
 color: red;  
}  
.heading{  
 font-size: 16px;  
 background-color: #ccc;  
}

* lúc này object style sẽ có dạng như sau:

styles {

heading: “Paragraph\_heading\_\_1kGV-“,

paragraph: Paragraph\_paragraph\_\_ThmR99”

}

Paragraph: Thư mục cha chứa file .module.css

heading: tên className

\_\_: ký tự ngăn cách

1kGV-: mã hóa đường dẫn hatch B64

* Để sử dụng className trong file js ta làm như sau:

import styles from ‘./App.module.css’

<p className = {styles.heading}>Hello</p>

* Tuy nhiên sẽ không sử dụng được các thẻ tag mặc định mà không gọi className, vì sẽ bị ảnh hưởng đến các file khác

VD:

h1{  
 color: green;  
}

Cách khắc phục là dùng cscc (đổi đuôi là App.module.scss), phải cài thư viện sass trước, đứng tại thư mục dự án (npm i sass)

.heading{  
 font-size: 16px;  
 background-color: #ccc;  
 h1{  
 color: green;  
 }  
}

* Nếu muốn dùng css chung thì nên tạo 1 file css riêng, không dùng index.css hay App.css

VD: GlobalStyles.css và sử dụng file js riêng biệt GlobalStyle.js

import './GlobalStyle.css'  
  
function GlobalStyle({children}) {  
 return {children}  
}  
export default GlobalStyle;

Rồi học file App.js lại bằng thẻ <GlobalStyle>

* Cách sử dụng nhiều class hoặc kết hợp class expression và class chuỗi, mỗi class cách nhau bằng dấu cách

VD:

import styles from './style.module.css'  
  
function App() {  
 return (  
 <div>  
 <h1 className={`${styles.heading} ${styles.content}`}>Hello</h1>  
 </div>  
 )  
}  
export default App;

hoặc

import styles from './style.module.css'  
  
function App() {  
 return (  
 <div>  
 <h1 className={[styles.heading, styles.content].join(' ')}>Hello</h1>  
 </div>  
 )  
}  
export default App;

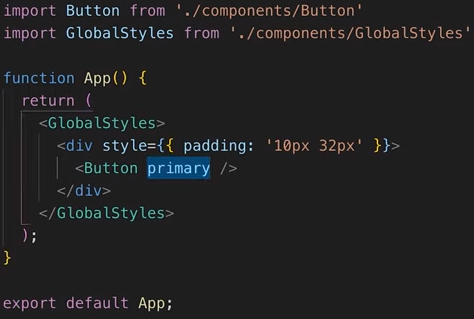
### Classes và thư viện clsx

Sử dụng clsx để

Cách cài, cd vào thư mục dự án: npm install clsx

import clsx from 'clsx'  
import styles from './style.module.css'  
  
function App() {  
 return (  
 <div>  
 <h1 className={clsx(styles.heading, styles.content)}>Hello</h1>  
 </div>  
 )  
}  
export default App;

Tại file index.js

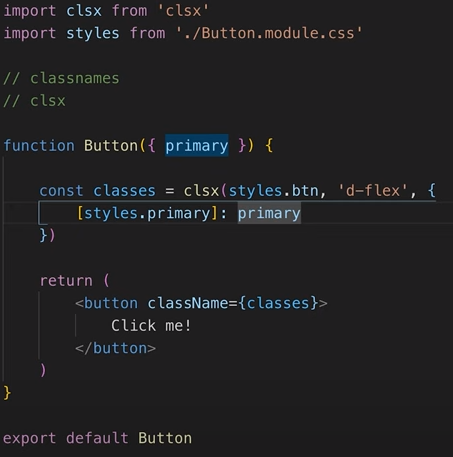


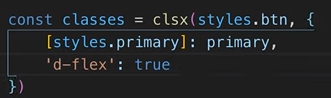
Tại file App.js



Khi truyền 1 object vào thì nó sẽ xét xem biến bên trái có không nếu có thì sẽ dùng còn không có thì không, bên phải sẽ nhận giá trị boolean quyết định xem có hiện không

Có thể tách clsx ra để tiện dùng





## React router V6, định hướng trong React

Là định tuyến di chuyển qua lại giữa các trang trong cùng một dự án, có thể xem thêm [tại đây](https://reactrouter.com/en/main).

Cách cài, tại thư mục dự án: npm install react-router-dom

Tại thư mục lớn nhất, index.js, bọc toàn bộ dự án (VD App.js)

import {BrowserRouter} from 'react-router-dom'

<BrowserRouter>  
 <App />  
</BrowserRouter>

Tạo pages gồm 3 trang js lần lượt là Home.js, News.js, Contact.js

function Home() {  
 return(  
 <h1>Home page</h1>  
 )  
}  
  
export default Home

Tại trang App.js

import { Routes, Route, ***Link***} from 'react-router-dom'  
import HomePage from './pages/Home'  
import NewsPage from './pages/News'  
import ContactPage from "./pages/Contact";  
  
function App() {  
 return(  
 <div className="app">  
 <nav>  
 <ul>  
 <li>  
 <Link to="/">Home</Link>  
 </li>  
 <li>  
 <Link to="/news">News</Link>  
 </li>  
 <li>  
 <Link to="/contact">Contact</Link>  
 </li>  
 </ul>  
 </nav>  
  
 <Routes>  
 <Route path="/" element={<HomePage />} />  
 <Route path="/news" element={<NewsPage />} />  
 <Route path="/contact" element={<ContactPage />} />  
 </Routes>  
 </div>  
 )  
}  
  
export default App

Giải thích:

* Sử dụng thẻ Routes để khai báo khu vực biến đổi, di chuyển qua lại, nếu chi di chuyển 1 phần còn các phần khác giữ nguyên thì không cho vào thẻ Routes
* Thẻ Route để định tuyến từng element của React
* Thẻ Link dùng thay thế thẻ a nếu là liên kết trong cùng dự án, còn nếu liên kết ngoài các website khác thì dùng thẻ a, điều này giúp việc di chuyển không bị load lại trang, trông mượt hơn (cơ chế e.preventDefault())

# Dự án Tiktok

## Các cài đặt cần thiết

### Tạo dự án tiktok

npx create-react-app tiktok

### Tạo ssh-key github (bỏ qua nếu có rồi)

trong phần github profile tại mục ssh-key nếu có rồi thì thôi còn chưa có thì tạo như sau

mở gitbatch gõ lệnh sau:

* B1: ls -al ~/.ssh

Nếu có ít nhất 1 file có đuôi .pub thì là đã có ssh-public rồi và chuyển tới bước B3

* B2: Tạo ssh key mới (thay bằng email github)

ssh-keygen -t ed25519 -C "your\_email@example.com"

nhấn enter để xác nhận nơi lưu ssh

nhập tạo pass xong nhấn enter

nhập lại pass rồi nhấn enter

gõ ls -al ~/.ssh lúc này đã có file .pub

* B3: nhấn cat ~/.ssh/tên file .pub, copy dòng mã hóa đó
* B4: Tại setting của SSH keys , nhấn new SSH key

title: github-ssh-key

key: dán đoạn mã hóa cop

nhấn tạo mới

### Tạo github repository

Tạo repo mới, chuyển sang tab ssh key

dán từng dòng đoạn 3 dòng cuối vào terminal

nhập passprase bên trên nhấn enter là xong

### Cài đặt customize-cra để tùy chỉnh webpack

Trong tương lai, sẽ có những cập nhật thay đổi webpack, tuy nhiên react đã tối ưu hóa file và không thể điều chỉnh, có thể dùng eject tuy nhiên không nên vì nó sẽ rất rối

Giải pháp là cài customize-cra, tại thư mục làm việc ...\tiktok>

npm install customize-cra react-app-rewired --save-dev

Check xem package.json nếu có dev-dependence thì đã được

Tại mục /tiktok tạo file config-overrides.js

/\* config-overrides.js \*/  
  
***module***.exports = function override(config, env) {  
 //do stuff with the webpack config...  
 return config;  
}

Trong tương lai nếu có cấu hình webpack thì viết vào đoạn giữa phần này

Tại mục packpage.json sửa scripts

start, build, test từ react-scripts start, build, test thành react-app-rewired start, build, test

npm start chạy thử xem có hoạt động không

### Cài đặt babel plugin module resolver

Nếu bạn cài đặt nhiều thư mục và chồng chất như này:

import MyUtilFn from '../../../../utils/MyUtilFn';

Bạn có thể dùng babel plugin module resolver để sửa thành như này:

import MyUtilFn from 'utils/MyUtilFn';

Để cài, vào thư mục dự án /tiktok> npm install --save-dev babel-plugin-module-resolver

Tại mục devDependensies trong package,json có:

"babel-plugin-module-resolver": "^5.0.0",

Tạo file /tiktok/.babelrc

{  
 "plugins": [  
 ["module-resolver", {  
 "alias": {  
 "~": "./src"  
 }  
 }]  
 ]  
}

Tạo file /tiktok/jsconfig.json

{  
 "compilerOptions": {  
 "baseUrl": ".",  
 "paths": {  
 "~/\*": ["src/\*"]  
 }  
 }  
}

Tại file config-overrides.js

const { override, useBabelRc } = ***require***("customize-cra");  
  
***module***.exports = override(  
 useBabelRc()  
);

### Cài đặt sass

npm i -D sass

npm i normalize.css

xóa bỏ các file không cần thiết: App.css, logo, index.css

## Xây dựng chương trình

### Tạo file style mặc định cho toàn trang

Tạo src/component/GlobalStyle/GlobalStyle.scss, index.js

index.js

import React from 'react';  
import ReactDOM from 'react-dom/client';  
import App from './App';  
import reportWebVitals from './reportWebVitals';  
import GlobalStyles from "~/component/GlobalStyle";  
  
const root = ReactDOM.createRoot(***document***.getElementById('root'));  
root.render(  
 <React.StrictMode>  
 <GlobalStyles>  
 <App />  
 </GlobalStyles>  
 </React.StrictMode>  
);  
  
// If you want to start measuring performance in your app, pass a function  
// to log results (for example: reportWebVitals(console.log))  
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals  
reportWebVitals();

GlobalStyle/index/js

import './GlobalStyles.scss';  
function GlobalStyle({children}) {  
 return children  
}  
  
export default GlobalStyle;

App.js

function App() {  
 return (  
 <div className="App">  
 <h1>React App</h1>  
 </div>  
 );  
}  
  
export default App;

### Tạo layout

npm i react-router-dom

Tạo src/pages/Home/index.js

import React from 'react';  
  
function Home() {  
 return (  
 <div>  
 <h2>Home Page</h2>  
 </div>  
 );  
}  
  
export default Home;

Tạo src/pages/Following/index.js

import React from 'react';  
  
function Following() {  
 return (  
 <div>  
 <h2>Following Page</h2>  
 </div>  
 );  
}  
  
export default Following;

Tại App.js

import {BrowserRouter, Routes, Route, ***Link***} from 'react-router-dom'  
import Home from '~/pages/Home'  
import Following from "~/pages/Following";  
  
function App() {  
 return (  
 <BrowserRouter>  
 <div className="App">  
 <Routes>  
 <Route path="/" element={<Home />} />  
 <Route path="/following" element={<Following />} />  
 </Routes>  
 </div>  
 </BrowserRouter>  
 );  
}  
  
export default App;

Đưa route ra ngoài, vì có thể sau này sẽ có rất nhiều route nên phân tách nó ra bên ngoài rồi import vào

Tạo một mục chứa route: src/routes/index.js

import Home from '~/pages/Home'  
import Following from "~/pages/Following";  
  
// Public Routes  
const ***publicRoutes*** = [  
 { path: '/', component: Home },  
 { path: '/following', component: Following }  
]  
  
// Private Routes  
const ***privateRoutes*** = [  
  
]  
  
export {***publicRoutes***, ***privateRoutes***}

Sửa App.js

import {BrowserRouter, Routes, Route, ***Link***} from 'react-router-dom'  
import {***publicRoutes***} from '~/routes'  
  
function App() {  
 return (  
 <BrowserRouter>  
 <div className="App">  
 <Routes>  
 {***publicRoutes***.map((route, index) => {  
 const Page = route.component  
 return(  
 <Route key={index} path={route.path} element={<Page />} />  
 )  
 })}  
 </Routes>  
 </div>  
 </BrowserRouter>  
 );  
}  
  
export default App;

# CÁC THƯ VIỆN KHÁC

## Font awesome

**Terminal:**

npm i --save @fortawesome/fontawesome-svg-core  
npm i --save @fortawesome/free-solid-svg-icons  
npm i --save @fortawesome/free-regular-svg-icons  
npm i --save @fortawesome/free-brands-svg-icons  
npm i @fortawesome/react-fontawesome

**Import:**

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'

## Bootrap

## Tailwind

**Terminal:**

npm install -D tailwindcss

npx tailwindcss init

**tailwind.config.js**

*/\*\* @type {import('tailwindcss').Config} \*/  
module*.exports = {  
 content: [  
 "./src/\*\*/\*.{js,ts,jsx,tsx}",  
 ],  
 theme: {  
 extend: {},  
 },  
 plugins: [],  
}

**input.css (nơi chứa file custom css)**

@tailwind base;

@tailwind components;

@tailwind utilities;

// your css

**Tạo file output.css (nơi chứa file css đã biên dịch – readonly)**

**Terminal:**

npx tailwindcss -i ./src/input.css -o ./src/output.css --watch

**image:**

className="bg-[url('../public/assets/images/ft.jpg')]"

**Import output.css vào file**

## Ant Design

npm install antd --save

## Create React App với typescript

npx create-react-app <YOUR-PROJECT-NAME> --template typescript

## Json server

**Terminal:**

npm install -g json-server

**Package.json**

"json-server": "json-server --watch ./src/data/db.json"

## React icon

npm install react-icons –save

import { FaBeer } from 'react-icons/fa';

# NGOÀI LỀ

## Cách căn chỉnh code đẹp

Bôi đen phần văn bản code nhấn Ctrl + Alt + L

## Cách đổi cổng chạy của React

vào file src của dự án bạn đang làm, tạo 1 file .env.local viết nội dung

PORT=X (với X là các cổng bạn muốn: 3000, 3001, ...)